Detection of Performance Changes in MooBench Results Using Nyrkiö on GitHub Actions

Shinhyung Yang[1], David Georg Reichelt[2], Henrik Ingo[3], and Wilhelm Hasselbring[1]

[1]Kiel University, Kiel, Germany
[2]Lancaster University Leipzig / URZ Leipzig, Leipzig, Germany
[3]Nyrkiö Oy, Järvenpää, Finland

Abstract

In GitHub with its 518 million hosted projects, performance changes within these projects are highly relevant to the project's users. Although performance measurement is supported by GitHub CI/CD, performance change detection is a challenging topic.

In this paper, we demonstrate how we incorporated Nyrkiö to MooBench. Prior to this work, Moobench continuously ran on GitHub virtual machines, measuring overhead of tracing agents, but without change detection. By adding the upload of the measurements to the Nyrkiö change detection service, we made it possible to detect performance changes. We identified one major performance regression and examined the performance change in depth. We report that (1) it is reproducible with GitHub actions, and (2) the performance regression is caused by a Linux Kernel version change.

1 Introduction

GitOps is a specific implementation and extension of DevOps practices, particularly focused on using Git as the single source of truth for infrastructure and application deployments. GitOps provides a set of development operations that embody the CI/CD tasks [7]. GitOps integrates DevOps operations of a software project with Git operations, e.g., with GitOps, a git-push action triggers a user-defined CI/CD pipeline, specific to the software project's repository.

GitHub has become the biggest GitOps platform for day-to-day development in open-source and enterprise projects. GitHub reports that it hosts 518 million projects, including one billion contributions. *GitHub Actions* is a CI/CD platform that includes both CI/CD pipelines and *GitHub runners*: Azure VM resources that run actions. The github-action-benchmark tool[1] is a continuous benchmarking tool for GitHub CI/CD; it is provided as a *GitHub action*, which receives the benchmarking data, and plots the result on a GitHub.io page. It is useful for comparing performance differences between git pushes, where the performance is not only impacted by software changes, but also by the VMs.

Benchmarks can be used for comparing different methods, techniques and tools [5], and MooBench [2, 6] is used for comparing tracing agents of monitoring frameworks such as Kieker [13].

In this paper, we demonstrate that performance change detection of MooBench's measurement results works by running the Nyriö change detection service, integrated with MooBench's GitHub actions workflow. Nyrkiö is a change detection service provided as a GitHub action. It uses the E-Divisive Means algorithm to detect changes [4]. In the examined versions, we see that the execution time of Kieker's AspectJ tracing agent increased significantly on GitHub's `ubuntu-latest` image, although the involved commits did not contain any changes to the software. We confirmed the detected change by reproducing and analyzing the performance change locally.

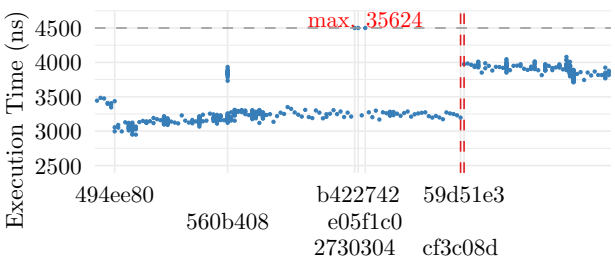
Our contributions for this paper are as follows: (1) we integrated the Nyrkiö change detection service to continouously analyse the MooBench benchmarking, (2) we discovered and replicated the performance change on GitHub runners, and (3) we configured GitHub-hosted and self-hosted runners for replicating the results.

2 Background

Continuous Benchmarking aims to continuously detect performance changes in the target system using performance measurements [3, 4]. MooBench is designed to continuously benchmark the overhead of tracing agents, initially on Jenkins [2] using a bare metal server, and extended to GitHub actions [9] using GitHub-hosted VMs. The github-action-benchmark is a plugin application native to the GitHub CI/CD, enabling collection and visualization of data from continuous benchmarking. Nyrkiö extends github-action-benchmark, incorporating the E-Divisive algorithm to detect performance changes, and utilizing the GitHub Issues board to notify important detection results.

E-Divisive Means Algorithm in Nyrkiö Matteson and James first introduced the E-Divisive al-

[1] `https://github.com/benchmark-action/github-action-benchmark`



Commit IDs (listed by date; from May 2024 to Aug 2025)

Figure 1: Our investigation focuses on the change between `cf3c08d` and `59d51e3` (--). Each data point represents the execution time of a Benchmark run. Values > 4500 ns are clipped and the maximum value 35 624 is caused by development code changes.

gorithm, which detects change points in the given numerical sequence [1]. We used Nyrkiö's E-Divisive implementation to validate that our initial finding on the performance change between two commits as shown in Figure 1. In Figure 2, produced by Nyrkiö, we see that our finding is indeed recognized as a performance change by E-Divisive Means: the diagram shows four change detections in four red dots, and the right-most dot on January 9, 2025 at 20:04 matches the performance change between `cf3c08d` and `59d51e3` in Figure 1. To produce it, we configured the p-value with 0.001, which finds fewer change points, decreasing false positives. Setting the change magnitude with 5 %, Nyrkiö only lists change points bigger than that.

3 Performance Change Examination

In this section, we present the performance change detection, our experiments for examining one change and the analysis of the experimental data.

3.1 Previously Detected Changes

In May 2024, MooBench started continuous benchmarking on the GitHub CI/CD [9]. In addition to git-push, GitHub actions allow for triggering a CI/CD pipeline with a scheduler to collect the data periodically. In March 2025, we noticed a performance regression of Kieker's AspectJ agent from the GitHub CI/CD results. The regression appeared between two benchmark runs triggered by two git-pushes, noted by two commit signatures, `cf3c08d` and `59d51e3`. We made those pushes to install the R package, which was pre-installed in the old `ubuntu-22.04` image, but obsoleted in the new `ubuntu-24.04` image, a decision by GitHub.[2] Our first intuition was that the performance regression was made by the Ubuntu version changes.

3.2 Experimental Setup

To replicate the performance change between two commit signatures `cf3c08d` and `59d51e3`, we deployed

[2] `https://github.com/actions/runner-images/issues/10636`

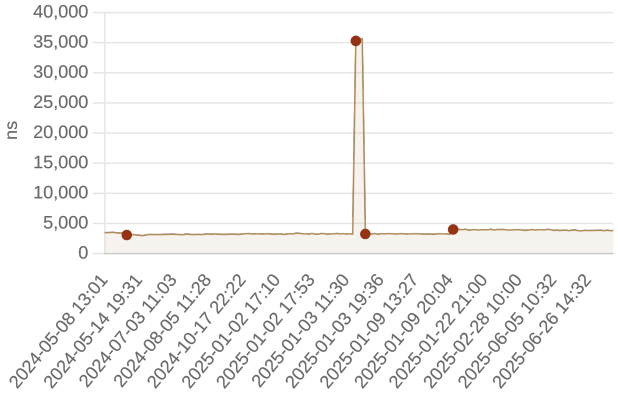


Figure 2: Nyrkiö's detected performance changes

eight runners in our GitHub Actions workflow: four GitHub-hosted runners and four self-hosted runners. All eight VMs were configured to start either by a git-push, or a scheduler that starts them every three hours.

GitHub-hosted runners: we use four standard GitHub-hosted runners. Each deploys a Linux image: two runners use the `ubuntu-22.04` image, and the other two use the `ubuntu-24.04` image. Kozlov reports that a standard GitHub-hosted runner has 4 vCPUs and 16 GiB RAM, which is equivalent to the `t4g.xlarge` instance on AWS [12].

Self-hosted runners: we use four self-hosted runners. Our self-hosted runners are four virtual machines with an identical spec; each has 4 vCPUs and 8 GiB RAM. The VMs belong to the same hosting server, which has 2 Xeon E5-2650 CPUs and 94 GiB RAM. Two of them use Ubuntu 22.04 and the other two use Ubuntu 24.04.

MooBench: We benchmarked the Kieker AspectJ agent, where we observed the performance change. MooBench uses its default configuration to run the Kieker AspectJ agent: one iteration measures the duration of a Java method, which recursively calls itself 10 times. One loop includes two million iterations, and the entire benchmark consists of 10 loops, and the system rests for 30 s at the end of each loop.

3.3 Statistical Analysis

Evaluation method: For all pairs of compared data series, we checked their normality using the Shapiro-Wilk normality test. The resulting w-value close to 1.00 confirms the normality of the compared data. We used the paired t-test to verify whether the difference of the two compared data series is statistically significant or not. The resulting p-value < 0.05 means the difference is significant.

Validation: We elaborated the results in Figure 3. First, we validated that the software change was not the cause of the performance change. The (w, p) between the two execution time series by commits `cf3c08d` and `59d51e3` are $(0.96, 0.07)$ (22.04/self),

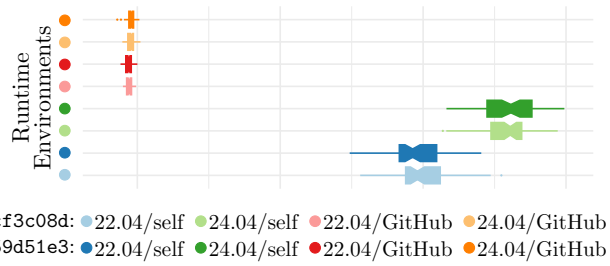


Figure 3: Measurements at commits `cf3c08d` and `59d51e3`. The boxplot shows the distribution of execution time measurements per runtime environment.

$(0.98, 0.65)$ (24.04/self), $(0.98, 0.65)$ (22.04/GitHub), and $(0.98, 0.49)$ (24.04/GitHub). Using our method, the compared data series are normally distributed, and the difference of comparisons are not significant. Second, we validated our assumption both on GitHub-hosted runners and self-hosted runners by comparing two execution time series by two Ubuntu versions 22.04 and 24.04: $(0.99, 0.001\,966\,284)$ (GitHub-hosted) and $\left(0.99, 1.368\,409 \times 10^{-50}\right)$ (self-hosted). In the future, we will investigate Ubuntu software changes and evaluate the differences, and also validate whether this is related to different kernel versions.

4 Related Work

The E-Divisive means algorithm was introduced in [1]. The efforts to integrate E-Divisive means algorithms to CI/CD has followed [4, 8, 11]. Regression testing in continuous integration is investigated in [3]. They utilize unit testing for regression testing, which compares two different code versions. Kozlov [12] did a comprehensive analysis on GitHub-hosted VMs and compared them to the equivalent AWS EC2 instances, and self-hosted VMs. Nyrkiö's GitHub action tool[3] is based on the github-action-benchmark tool. Extending the MooBench CI/CD to GitHub and the use of github-action-benchmark discussed in [10].

5 Conclusion and Future Work

In this work, we discussed how to continuously detect performance changes for MooBench. Using the Nyrkiö change detection, we identified a significant performance change. To show the performance change does not only occur on GitHub-hosted VMs, we replicated the performance change on self-hosted VMs in our local servers too. In the future, we will further investigate on the performance change, and use the updated MooBench GitHub workflow to examine regressions caused by source code changes of the tracing agents.

Acknowledgment This research is funded by the Deutsche Forschungsgemeinschaft (DFG – German Research Foundation), grant no. 528713834.

[3] `https://github.com/nyrkio/github-action-benchmark`

References

[1] D. S. Matteson and N. A. James. "A Nonparametric Approach for Multiple Change Point Analysis of Multivariate Data". In: *Journal of the American Statistical Association* 109.505 (2014). DOI: 10.1080/01621459.2013.849605.

[2] J. Waller, N. C. Ehmke, and W. Hasselbring. "Including Performance Benchmarks into Continuous Integration to Enable DevOps". In: *SIGSOFT Softw. Eng. Notes* 40.2 (2015). DOI: 10.1145/2735399.2735416.

[3] D. G. Reichelt, S. Kühne, and W. Hasselbring. "PeASS: A Tool for Identifying Performance Changes at Code Level". In: ASE '19. 2019. DOI: 10.1109/ASE.2019.00123.

[4] D. Daly et al. "The Use of Change Point Detection to Identify Software Performance Regressions in a Continuous Integration System". In: ICPE. 2020. DOI: 10.1145/3358960.3375791.

[5] W. Hasselbring. "Benchmarking as Empirical Standard in Software Engineering Research". In: *EASE*. 2021. DOI: 10.1145/3463274.3463361.

[6] D. G. Reichelt, S. Kühne, and W. Hasselbring. "Overhead Comparison of OpenTelemetry, inspectIT and Kieker". In: SSP '21. PID: Vol-3043. CEUR Workshop Proceedings, 2021.

[7] F. Beetz and S. Harrer. "GitOps: The Evolution of DevOps?" In: *IEEE Software* 39.4 (2022). DOI: 10.1109/MS.2021.3119106.

[8] M. Fleming et al. "Hunter: Using Change Point Detection to Hunt for Performance Regressions". In: ICPE '23. ACM, 2023. DOI: 10.1145/3578244.3583719.

[9] D. G. Reichelt, R. Jung, and A. van Hoorn. "Overhead Measurement Noise in Different Runtime Environments". In: SSP '24. PID: 20.500.12116/45533. 2024.

[10] D. G. Reichelt et al. "Overhead Comparison of Instrumentation Frameworks". In: ICPE '24. 2024. DOI: 10.1145/3629527.3652269.

[11] H. Ingo. *8 Years of Optimizing Apache Otava: How disconnected open source developers took an algorithm from n3 to constant time*. 2025. DOI: 10.48550/arXiv.2505.06758.

[12] I. Kozlov. "A comparative study of GitHub-hosted, self-hosted, and Kubernetes-based GitHub Runners for web applications GitHub Actions workflows". PID: 10024/882579. 2025.

[13] S. Yang et al. "The Kieker Observability Framework Version 2". In: ICPE '25 Companion. 2025. DOI: 10.1145/3680256.3721972.